

```
Attribute VB_Name = "shapes"
Option Explicit
```

```
Public Type ContourSegment
```

```
    x As Double
    y As Double
    len As Double
    udx As Double
    udy As Double
```

```
End Type
```

```
Public Type ContourList
```

```
    segments() As ContourSegment
End Type
```

```
Public Type ContourSet
```

```
    lists() As ContourList
    perimeter As Double
    centerOfProjectionX As Double
    centerOfProjectionY As Double
    sizeMerit As Double
    angleMerit As Double
    translationMerit(1) As Double
    translationAngle(1) As Double
```

```
End Type
```

```
Public Sub render(shape As Object, img As Bmp, rp As RenderProfile)
```

```
    Dim i As Integer, j As Integer
    Dim x0 As Double, y0 As Double
```

```
    x0 = img.width / 2 + 0.5
    y0 = img.height / 2 - 0.5
    For j = 0 To img.height - 1
        For i = 0 To img.width - 1
            img.Pixel(i, j) = rp.PixelValue(shape.profileCoord(i - x0, y0 - j))
        Next i
    Next j
```

```
End Sub
```

```
Public Sub scanContours(shape As Object, step As Double, points As ContourSet)
```

```
    Const gate = 0.1 ' radians
```

```
Const maxSteps = 100000
Const block = 100
```

```
Dim ci As Integer, cn As Integer
Dim si As Long, pn As Long, done As Boolean
Dim x As Double, y As Double, dx As Double, dy As Double, x0 As Double, y0 As Double
Dim xn As Double, yn As Double, xp As Double, yp As Double
Dim ap As Double, an As Double, r As Double
Dim u As Double, px As Double, py As Double
Dim gt As Double, cs As Double, sn As Double
Dim perimeter As Double
```

```
cn = shape.numContours
Redim points.lists(cn - 1)
```

```
For ci = 0 To cn - 1
    ' get starting point, direction
    shape.getContourStart ci, x0, y0
    x = x0
    y = y0
    ap = shape.profileCoord(x + step, y)
    an = shape.profileCoord(x, y + step)
    r = Sqr(ap ^ 2 + an ^ 2)
    dx = an / r
    dy = -ap / r
```

```
si = 0
Redim points.lists(ci).segments(block - 1)
done = False
```

```
Do While Not done And si < maxSteps
    ' rotate gate so contour passes thru
    gt = 0.5 * gate
    an = 0
    Do While True
        cs = Cos(gt)
        sn = Sin(gt)
        xp = x + step * (dx * cs - dy * sn)
        yp = y + step * (dx * sn + dy * cs)
        ap = shape.profileCoord(xp, yp)
        If ap >= 0 Then Exit Do
```

```

xn = xp
yn = yp
an = ap
gt = gt + gate
If gt >= pi Then Err.Raise vbObjectError + 200, , "Lost contour CCW at (" & Format(x, "0.00") &
, " & Format(y, "0.00") & ") after " & si & " steps"
pn = pn + 1
Loop
If an = 0 Then
Do While True
gt = gt - gate
If gt <= -pi Then Err.Raise vbObjectError + 200, , "Lost contour CW at (" & Format(x, "0.00")
& ", " & Format(y, "0.00") & ") after " & si & " steps"
cs = Cos(gt)
sn = Sin(gt)
xn = x + step * (dx * cs - dy * sn)
yn = y + step * (dx * sn + dy * cs)
an = shape.profileCoord(xn, yn)
If an < 0 Then Exit Do
xp = xn
yp = yn
ap = an
pn = pn + 1
Loop
End If

' get next point, distance, direction
xp = xp + (xn - xp) * ap / (ap - an)
yp = yp + (yn - yp) * ap / (ap - an)
r = Sqr((xp - x) ^ 2 + (yp - y) ^ 2)
dx = (xp - x) / r
dy = (yp - y) / r

' test for done
px = x0 - x
py = y0 - y
u = px * dx + py * dy
If 0 < u And u <= 1.01 * r Then
px = px - u * dx
py = py - u * dy
If Sqr(px ^ 2 + py ^ 2) <= step / 2 Then

```

```

xp = x0
yp = y0
r = Sqr((xp - x) ^ 2 + (yp - y) ^ 2)
dx = (xp - x) / r
dy = (yp - y) / r
done = True
End If
End If

' write next point
If si > UBound(points.lists(ci).segments) Then
    Redim Preserve points.lists(ci).segments(si - 1 + block)
End If

With points.lists(ci).segments(si)
    .x = x
    .y = y
    .len = r
    .udx = dx
    .udy = dy
End With

' update stats
perimeter = perimeter + r
si = si + 1
x = xp
y = yp
Loop
Redim Preserve points.lists(ci).segments(si - 1)

If Not done Then
    ' logic error in algorithm
    Err.Raise vbObjectError + 201, , "Can't find end of contour"
End If
Next ci

points.perimeter = perimeter
pn = pn + 2 * si + 2
End Sub

Public Sub computeMerit(contours As ContourSet)

```

```

Dim ci As Long, si As Long, d As Integer
Dim x As Double, y As Double, r As Double, t As Double, cs As Double, sn As Double
Dim Src As Double, Srs As Double, Ss2 As Double, Sc2 As Double, Scs As Double
Dim Sdot As Double, Scross As Double
Dim Strans(179) As Double, cstab(179) As Double, snTab(179) As Double

' center of projection
For ci = 0 To UBound(contours.lists)
    For si = 0 To UBound(contours.lists(ci).segments)
        With contours.lists(ci).segments(si)
            cs = .udy
            sn = -.udx
            r = (.x + .udx * .len / 2) * cs + (.y + .udy * .len / 2) * sn
            Src = Src + .len * r * cs
            Srs = Srs + .len * r * sn
            Ss2 = Ss2 + .len * sn * sn
            Sc2 = Sc2 + .len * cs * cs
            Scs = Scs + .len * cs * sn
        End With
    Next si
Next ci
r = Sc2 * Ss2 - Scs ^ 2
contours.centerOfProjectionX = (Src * Ss2 - Srs * Scs) / r
contours.centerOfProjectionY = (Srs * Sc2 - Src * Scs) / r

' compute sine and cosine tables
For d = 0 To 179
    t = d * degToRad
    cstab(d) = Cos(t)
    snTab(d) = Sin(t)
Next d

' figures of merit
For ci = 0 To UBound(contours.lists)
    For si = 0 To UBound(contours.lists(ci).segments)
        With contours.lists(ci).segments(si)
            ' angle and size
            x = .x + .udx * .len / 2 - contours.centerOfProjectionX
            y = .y + .udy * .len / 2 - contours.centerOfProjectionY
            Sdot = Sdot + .len * Abs(x * .udy - y * .udx)
            Scross = Scross + .len * Abs(x * .udx + y * .udy)
        End With
    Next si
Next ci

```

```

' translation
For d = 0 To 179
  Strans(d) = Strans(d) + .len * Abs(csTab(d) * .udy - snTab(d) * .udx)
Next d
End With
Next si
Next ci
contours.angleMerit = Scross
contours.sizeMerit = Sdot

' translation figures of merit
For d = 0 To 1
  contours.translationMerit(d) = Strans(0)
  contours.translationAngle(d) = 0
Next d
For d = 1 To 179
  If Strans(d) < contours.translationMerit(0) Then
    contours.translationMerit(0) = Strans(d)
    contours.translationAngle(0) = d
  ElseIf Strans(d) > contours.translationMerit(1) Then
    contours.translationMerit(1) = Strans(d)
    contours.translationAngle(1) = d
  End If
Next d
End Sub

Public Sub drawContours(cs As ContourSet, dsply As Display)
  Dim ci As Long, si As Long
  Dim tblt As New tablet
  Dim x1 As Double, y1 As Double

  For ci = 0 To UBound(cs.lists)
    x1 = cs.lists(ci).segments(UBound(cs.lists(ci).segments)).x
    y1 = cs.lists(ci).segments(UBound(cs.lists(ci).segments)).y
    For si = 0 To UBound(cs.lists(ci).segments)
      With cs.lists(ci).segments(si)
        tblt.DrawLine x1, y1, .x, .y, vbGreen
        x1 = .x
        y1 = .y
      End With
    Next si
  Next ci
End Sub

```

Next si
Next ci

dsply.Drawsketch tblt, ClientCoordinates
End Sub

Attribute VB_Name = "vbmath"
Option Explicit

Public Const pi = 3.14159265358979
Public Const twopi = 2 * pi
Public Const halfpi = pi / 2
Public Const degToRad = twopi / 360

Public Function atan2(y As Double, x As Double) As Double
If x = 0 And y = 0 Then Exit Function
If y > x Then

If -y > x Then
 ' 135 - 225
 atan2 = pi - Atn(y / -x)
Else
 ' 45 - 135
 atan2 = halfpi - Atn(x / y)
End If

Else
If -y > x Then
 ' 225 - 315
 atan2 = 1.5 * pi + Atn(x / -y)
Elseif y >= 0 Then
 ' 0 - 45
 atan2 = Atn(y / x)
Else
 ' 315 - 0
 atan2 = twopi + Atn(y / x)
End If

End If
End Function

Public Function rmod(x As Double, m As Double) As Double
rmod = x - Int(x / m) * m
End Function

```

Public Function smod(x As Double, m As Double) As Double
    smod = rmod(x + m / 2, m) - m / 2
End Function

Public Function splitmod(lo As Double, hi As Double, m As Double) As Double
    splitmod = rmod(lo + smod(hi - lo, m) / 2, m)
End Function

Public Function min(a As Variant, b As Variant) As Variant
    If a < b Then min = a Else min = b
End Function

Public Function max(a As Variant, b As Variant) As Variant
    If a > b Then max = a Else max = b
End Function

Public Function limit(x As Variant, lo As Variant, hi As Variant) As Variant
    If x <= lo Then
        limit = lo
    ElseIf x >= hi Then
        limit = hi
    Else
        limit = x
    End If
End Function

VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
    Persistable = 0 'NotPersistable
    DataBindingBehavior = 0 'vbnNone
    DataSourceBehavior = 0 'vbnNone
    MTSTransactionMode = 0 'NotAnMTSObject
END
Attribute VB_Name = "RenderProfile"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

```



```
Public sigmoid As Double
Public background As Double
Public contrast As Double
Public noise As Double
```

```
Public Function realIntensity(profileCoord As Double) As Double
    Dim a As Double
```

```
    If sigmoid = 0 Then
        realIntensity = background + (Sgn(profileCoord) + 1) / 2 * contrast
    Else
        a = profileCoord / sigmoid
        If a <= -20 Then
            realIntensity = background
        ElseIf a >= 20 Then
            realIntensity = background + contrast
        Else
            realIntensity = background + contrast / (1# + Exp(-a))
        End If
    End If
```

```
End Function
```

```
Public Function pixelValue(profileCoord As Double) As Byte
```

```
    Dim z As Double
    z = realIntensity(profileCoord) + 0.5
    If z < 0 Then z = 0
    If z > 255 Then z = 255
    pixelValue = z
```

```
End Function
```

```
VERSION 1.0 CLASS
```

```
BEGIN
```

```
    MultiUse = -1 'True
    Persistable = 0 'NotPersistable
    DataBindingBehavior = 0 'vNone
    DataSourceBehavior = 0 'vNone
    MTSTransactionMode = 0 'NotAnMTSObject
```

```
END
```

```
Attribute VB_Name = "FanShape"
```

```
Attribute VB_GlobalNameSpace = False
```

```
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit
```

```
Const bladeNum = 4
```

```
Private Type BladeParams
    azimuth As Double      ' start of blade at outer radius, degrees
    width As Double        ' width of blade, degrees
    skew As Double         ' azimuth difference, outer to origin, degrees
    spiral As Double       ' center to edge radius difference, fraction of outer radius
End Type
```

```
Private Type FanParams
    outerRad As Double
    innerRad As Double
    round As Double
    blades(bladeNum - 1) As BladeParams
    holeInnerRad As Double
    holeOuterRad As Double
    holeThickness As Double
End Type
```

```
Private Type BladedData
    low As Double
    high As Double
    zoneLow As Double
    lowSkew As Double
    zoneWidth As Double
    widthSkew As Double
    center As Double
    halfWidth As Double
    skew As Double
    spiral As Double
    holeZoneT As Double
    holeRadT As Double
    hubRound As Double
    bladeRound As Double
    holeInnerRound As Double
    holeOuterRound As Double
```

End Type

Private params As FanParams
Private paramsChanged As Boolean

Private outerRad As Double
Private innerRad As Double
Private blades(bladeNum - 1) As BladeData
Private holeZoneRLow As Double
Private holeZoneRHigh As Double
Private holeCenterR As Double
Private holeRadR As Double

Private Sub loadFanData(fp As FanParams)
Dim k As Integer, knext As Integer, kprev As Integer

With fp

outerRad = .outerRad
innerRad = .innerRad * outerRad

holeZoneRLow = .holeInnerRad / 2 * outerRad
holeZoneRHigh = (.holeOuterRad + 1) / 2 * outerRad
holeCenterR = (.holeInnerRad + .holeOuterRad) / 2 * outerRad
holeRadR = (.holeOuterRad - .holeInnerRad) / 2 * outerRad

For k = 0 To bladeNum - 1

With blades(k)

.skew = fp.blades(k).skew
.low = fp.blades(k).azimuth - .skew
.high = fp.blades(k).width + .low
.halfwidth = fp.blades(k).width * degToRad / 2
.center = rmod(.low * degToRad + .halfwidth, twopi)
.skew = .skew / outerRad * degToRad
.spiral = fp.blades(k).spiral / .halfwidth * outerRad
.holeRadR = fp.holeThickness * .halfwidth
If .holeRadR > 0 Then
.holeZoneR = (.holeRadR + .halfwidth) / 2
Else
.holeZoneR = 0
End If
.bladeRound = min(fp.round, min((outerRad - innerRad) / 2, .halfwidth * outerRad))

```

        .holeInnerRad = min(fp.round, min(holeRadR, .holeRadT * fp.holeInnerRad * outerRad))
        .holeOuterRad = min(fp.round, min(holeRadR, .holeRadT * fp.holeOuterRad * outerRad))
    End With
Next k

    For k = 0 To bladeNum - 1
        With blades(k)
            knext = (k + 1) Mod bladeNum
            kprev = (k - 1 + bladeNum) Mod bladeNum
            .zoneLow = splitmod(blades(kprev).high, .low, 360) * degToRad
            .lowSkew = (.skew + blades(kprev).skew) / 2
            .zoneWidth = rmod(splitmod(.high, blades(knext).low, 360) * degToRad - .zoneLow, twopi)
            .widthSkew = (.skew + blades(knext).skew) / 2 - .lowSkew
            .hubRound = min(fp.round, min((outerRad - innerRad) / 2, max(smod((.center - .halfwidth +
                innerRad * .skew) - (blades(kprev).center + blades(kprev).halfwidth + innerRad * blades(kprev).skew), twopi), 0)
                * innerRad / 2))
            End With
        Next k
    End With
End Sub

Private Sub getFandData()
    If paramsChanged Then
        loadFandData params
        paramsChanged = False
    End If
End Sub

Public Function profileCoord(x As Double, y As Double) As Double
    Dim k As Integer
    Dim r As Double, t As Double, u As Double, v As Double, round As Double
    Dim rh As Double, rp As Double, a As Double

    getFandData

    r = Sqr(x ^ 2 + y ^ 2)
    t = atan2(y, x)

    ' figure out which blade to make
    For k = 0 To bladeNum - 1
        With blades(k)

```

```

      If rmod(t - (.zoneLow + r * .lowSkew), twopi) < .zoneWidth + r * .widthSkew Then Exit For
    End With
  Next k
  If k = bladeNum Then Err.Raise vbObjectError + 100, , "Bad blade detection logic"

  With blades(k)
    ' get blade relative azimuth
    t = t - r * .skew
    t = smod(t - .center, twopi)
    rp = r - t * .spiral
    If t >= 0 Then
      round = blades(k + 1) Mod bladeNum).hubRound
    Else
      t = -t
      round = .hubRound
    End If

    ' determine zone and distance from edge
    ' "<" instead of "<=" so hole can be disabled
    If holeZoneLow < rp And rp < holeZoneHigh And t < .holeZone Then
      rh = rp - holeCenter
      If rh >= 0 Then
        round = .holeOuterRound
      Else
        rh = -rh
        round = .holeInnerRound
      End If
      u = rh - (holeRadR - round)
      v = r * t - (r * .holeRadT - round)
      If u > 0 And v > 0 Then
        a = Sqr(u ^ 2 + v ^ 2) - round
      Elseif rh - holeRadR >= r * (t - .holeRadT) Then
        a = rh - holeRadR
      Else
        a = r * (t - .holeRadT)
      End If
    Else
      u = rp - (outerRad - .bladeRound)
      v = r * t - (r * .halfWidth - .bladeRound)
      If u > 0 And v > 0 Then
        a = .bladeRound - Sqr(u ^ 2 + v ^ 2)
      End If
    End If
  End With

```

```

Else
    u = r - (innerRad + round)
    v = r * t - (r * .halfwidth + round)
    If u < 0 And v < 0 Then
        a = Sqr(u ^ 2 + v ^ 2) - round
    Elseif rp - outerRad >= r * (t - .halfwidth) Then
        a = outerRad - rp
    Elseif r - innerRad >= r * (t - .halfwidth) Then
        a = r * (.halfwidth - t)
    Else
        a = innerRad - r
    End If
End If
End With

profileCoord = a
End Function

Public Sub render(img As Bmp, rp As RenderProfile)
    Dim i As Integer, j As Integer
    Dim x0 As Double, y0 As Double

    x0 = img.width / 2 + 0.5
    y0 = img.Height / 2 - 0.5
    For j = 0 To img.Height - 1
        For i = 0 To img.width - 1
            img.Pixel(i, j) = rp.pixelValue(profileCoord(i - x0, y0 - j))
        Next i
    Next j
End Sub

Public Property Get numBlades() As Variant
    numBlades = bladenNum
End Property

Public Property Get outerRadius() As Double
    outerRadius = params.outerRad
End Property

Public Property Let outerRadius(ByVal vNewValue As Double)

```

```
params.outerRad = max(vNewValue, 0)
paramsChanged = True
End Property
```

```
Public Property Get innerRadius() As Double
Attribute innerRadius.VB_Description = "Radius of fan hub, fraction of outer radius"
innerRadius = params.innerRad
End Property
```

```
Public Property Let innerRadius(ByVal vNewValue As Double)
params.innerRad = limit(vNewValue, 0, 1)
paramsChanged = True
End Property
```

```
Public Property Get cornerRound() As Double
Attribute cornerRound.VB_Description = "Corner radius, absolute units"
cornerRound = params.round
End Property
```

```
Public Property Let cornerRound(ByVal vNewValue As Double)
params.round = max(vNewValue, 0)
paramsChanged = True
End Property
```

```
Public Property Get azimuth(bn As Integer) As Double
Attribute azimuth.VB_Description = "Azimuth of blade bn in degrees"
azimuth = params.blades(bn).azimuth
End Property
```

```
Public Property Let azimuth(bn As Integer, ByVal vNewValue As Double)
params.blades(bn).azimuth = rmod(vNewValue, 360)
paramsChanged = True
End Property
```

```
Public Property Get width(bn As Integer) As Double
Attribute width.VB_Description = "Width of blade bn in degrees"
width = params.blades(bn).width
End Property
```

```
Public Property Let width(bn As Integer, ByVal vNewValue As Double)
params.blades(bn).width = limit(vNewValue, 0, 360)
```

```

        paramsChanged = True
    End Property

    Public Property Get skew(bn As Integer) As Double
        Attribute skew.VB_Description = "Blade skew (hub to circumference offset) in degrees"
        skew = params.blades(bn).skew
    End Property

    Public Property Let skew(bn As Integer, ByVal vNewValue As Double)
        params.blades(bn).skew = limit(vNewValue, -90, 90)
        paramsChanged = True
    End Property

    Public Property Get spiral(bn As Integer) As Double
        Attribute spiral.VB_Description = "Outer blade edge radial difference"
        spiral = params.blades(bn).spiral
    End Property

    Public Property Let spiral(bn As Integer, ByVal vNewValue As Double)
        params.blades(bn).spiral = limit(vNewValue, -0.5, 0.5)
        paramsChanged = True
    End Property

    Public Property Get holeInnerRadius() As Double
        Attribute holeInnerRadius.VB_Description = "Fraction of outerRadius"
        holeInnerRadius = params.holeInnerRad
    End Property

    Public Property Let holeInnerRadius(ByVal vNewValue As Double)
        params.holeInnerRad = limit(vNewValue, 0, 1)
        paramsChanged = True
    End Property

    Public Property Get holeOuterRadius() As Double
        Attribute holeOuterRadius.VB_Description = "Fraction of outerRadius"
        holeOuterRadius = params.holeOuterRad
    End Property

    Public Property Let holeOuterRadius(ByVal vNewValue As Double)
        params.holeOuterRad = limit(vNewValue, 0, 1)
        paramsChanged = True
    End Property

```


End Property

Public Property Get holeWidth() As Double
 Attribute holeWidth.VB_Description = "Width of holes, fraction of blade widths"
 holeWidth = params.holeThickness
 End Property

Public Property Let holeWidth(ByVal vNewValue As Double)
 params.holeThickness = limit(vNewValue, 0, 1)
 paramsChanged = True
 End Property

Public Property Get numContours() As Integer
 If params.holeThickness > 0 Then
 numContours = 5
 Else
 numContours = 1
 End If
 End Property

Public Sub getContourStart(i As Integer, x As Double, y As Double)
 Dim r As Double, t As Double, k As Integer

getFanData
 Select Case i

Case 0
 r = outerRad
 k = 0

Case 1 To 4
 r = holeCenterR + holeRadR
 k = i - 1
 End Select

t = blades(k).center + r * blades(k).skew
 x = r * Cos(t)
 y = r * Sin(t)
 End Sub

Private Sub Class_Initialize()
 Dim i As Integer

```

With params
  .outerRad = 128
  .innerRad = 0.2
  .round = 6
  .holeInnerRad = 0.5
  .holeOuterRad = 0.8
  .holeThickness = 0.33
  For i = 0 To bladenum - 1
    With .blades(i)
      .azimuth = (i + .0.25) * 360 / bladenum
      .width = 360# / (2 * bladenum)
      .skew = 0
      .spiral = 0
    End With
  Next i
End With
paramsChanged = True
End Sub

```